Assignment 2 solutions

1 a)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
final table	96	51	36	83	21	8			25		27	45		49	99	32	
collision attempts	0			49					8			96			96	83	
									49							49	

$h_1(45) = 11$

$h_1(99) = 14$

$h_1(32) = 15$

$h_1(96) = 11$, $h_2(96) = 3. \Rightarrow$ slot 0

$h_1(25) = 8$

$h_1(36) = 2$

$h_1(83) = 15$, $h_2(83) = 5 \Rightarrow$ slot 8

$h_1(27) = 10$

$h_1(21) = 4$

$h_1(49) = 15$ $h_2(49) = 5 \Rightarrow$ slot 13

$h_1(51) = 0$ $h_2(0) = 1 \Rightarrow$ slot 1.

$h_1(8) = 8$ $h_2(8) = 14 \Rightarrow$ slot 5.

* Note: if you had version with 2nd key 32, you would have 32 in slots 13 and 15.

* Slightly different table below if you had 2nd key 32.

b)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
final table	51		36	83	21		49		25		27	45		8	99	32	96
collisions				49					8			11			49	83	49
																49	

$h(45) = 11$

$h(99) = 14$

$h(32) = 15$

$h(96) = 11$ $h(96,1) = 16$

$h(25) = 8$

$h(36) = 2$

$h(83) = 15$, $h(83,1) = 3$.

$h(27) = 10$

$h(21) = 4$

$h(49) = 15$ $h(49,1) = 3$. $h(49,2) = 14$

$h(49,3) = 14$, $h(49,4) = 3$, $h(49,5) = 15$

$h(49,6) = 16$, $h(49,7) = 6$ ...

$h(51) = 0$

$h(8) = 8$ $h(8,1) = 13$

45, 99, 32, 96, , 25, 36, 83, 27, 21, 49, 51, 8.

c) Hash function: $k^2 \bmod 17$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
51		96 – 36		32 – 83 – 49				45	99				25 – 8		27	21

d) Table 1) The insertion of key 15 required 6 probes: collisions same as for '49': 15, 3, 8, 13, 1, and inserts at 6

Table 2) The insertion of 15 required 11 probes: collisions at 15, 3, 14, 14, 3, 15, 16, 6, 2, 4, and inserts at 12

Table 3) The insertion of 15 would be inserted onto the chain in slot 4. This requires only 1 probe, and 15 can be added to this list in constant time (using a reference to the front/back of list).

e) Given a "jump" value of $h_2(K)$, then since the table has size 17, the probe sequence will repeat every 17 probes.

ex. $h_1 = 1$ $h_2 = 3$

Probe sequence: $\underbrace{1, 4, 7, 10, 13, 16, 2, 5, 8, 11, 14, 0, 3, 6, 9, 12, 15}$, 1, 4, ...

all slots checked in 17 probes, since h_2 is relatively prime to 17.

∴ After 17 probes, if no free spot is found, we conclude the table is full.

Cases where double hashing fails to search entire table: (ie: fails to find free spot).

1) if $h_2 = 0$, the double hashing will not search for a free spot.

2) if $h_2(K)$ is not relatively prime to the table size, the probe sequence will repeat searching the same spots, and not search the entire table.

25, 37, 52, 14, 89, 35, 83, 53, 31, 86, 99, 46, 66, 34, 22, 2, 8, 90, 30, 68,

21, 17, 84, 29, 77, 45, 33, 41, 19, 53, 42, 93, 23, 18, 91

2a) 1) Create Sorted Groups of size 5

K=14

14	31	22	2	17	19	18
25	35	34	8	21	33	23
37	53	46	30	29	41	42
52	83	66	68	77	45	91
89	86	99	90	84	53	93

← Recursive call with K=4 to find median of medians.

2) Find med. of medians recursively:

29

30

37 41 ← input ≤ 5 .

46 42 ∴ Return 37

53.

Partition about 37:

29 30 37 $\underbrace{46\ 53\ 41\ 42}$

rank 3

Recursive call wi

Recursive call on 46 53 41 42 with K=1

Since input ≤ 5, return 41.

∴ med. of medians is 41

3) Partition input about 41:

25	37	14	35	31		52	89	83	53	86
34	22	2	8	30	41	99	46	66	90	68
21	17	29	33	19	r=17	84	77	45	53	42
23	18					93	91			

Recurse with K=14

4) Create sorted groups of size 5:

14	2	17	
25	8	19	
31	22	21	18
35	30	29	23
37	34	33	

← find item of rank 2. Since ≤ 5 items, return 21

Partition about 21:

14 2 8 17 19 18 21 $\underbrace{25\ 37\ 35\ 31\ 34\ 22\ 30\ 29\ 33\ 23}_{\text{Recurse } K=7}$

rank=7

5) Create sorted lists of size 5, K=7

25	22
31	23
34	29
35	30
37	33

← Return 29

Partition about 29:

25 22 23 29 37 $\underbrace{35\ 31\ 34\ 30\ 33}_{\text{Recurse } K=3.}$

rank=4

6) Create sorted groups of size 5:

30	
31	
34	33
35	
37	

← Return 33.

Partition about 33:

31 30 33 37 35 34

rank = K
3

∴ Return 33

2b) 1) Pivot 25, K=14

14 22 2 8 21 17 19 25 37 52 89 35 83 53 31 86 99 46 66 34 90 30 68
23 18 84 29 77 45 33 41 53 42 93 91

rank = 10

Recurse, K=4

2) Pivot 37, K=4

$\underbrace{35\ 31\ 34\ 30\ 29\ 33}_{\text{Recurse, } K=4}$ 37 52 89 83 53 86 99 46 66 90 68 84 77 45
41 53 42 93 91

rank 7

3) Pivot 35, K=4

$\underbrace{31\ 34\ 30\ 29\ 33}_{\text{Recurse } K=4}$ 35 Rank 6

4) Pivot 31, K=4

30 29 31 34 $\underbrace{33}_{}$. Recurse K=1

rank = 3

5) Pivot 34 K=1

33 34 rank = 2

Recurse K=1

6) Pivot 33 K=1

33 rank 1

∴ Return element 33.

2c) Step 1) Partition input into groups of size 6 : $\Theta(n)$

Step 2) Use insertion sort to sort each group. One group of 6 elements is sorted in constant time. Number of groups is $n/6$. $\therefore$ Total runtime is $\Theta(n)$

Step 3) Use 3rd element of each group as median. Make a recursive call using medians as input, and use new rank of $\frac{(n/6)}{2} \cong \frac{n}{12}$.
$T(n/6)$. Returned element is called x.

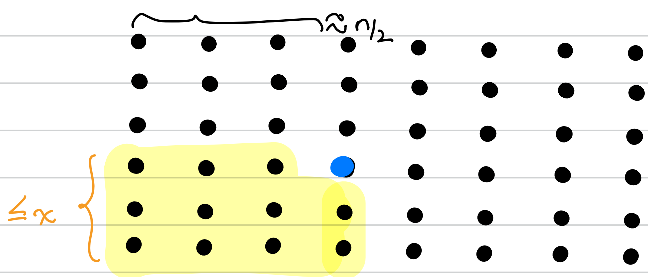
Step 4) Partition original input into a set L of items $< x$, and R of items $> x$.

Step 5) Let r be the rank of x.

If $k < r$, call Select using set L and rank k.

If $k > r$, call Select using set R and rank $k-r$.

\# of elements in L is



Runtime:

\# elements in L is at least $(n/2)\cdot(3/6) \cong n/4$

$\therefore |L| \geq n/4$

$\therefore |R| \leq \frac{3n}{4}$

$\therefore \leq T(3n/4)$

Total Runtime: $T(n) \leq \underbrace{T(n/6)}_{\text{step 3}} + \underbrace{T(3n/4)}_{\text{step 5}} + \underbrace{cn}_{\text{linear steps.}}$

Show $T(n) \leq dn$ by substitution:

Assume: $T(n/6) \leq dn/6 \qquad T(3n/4) \leq d(3n/4)$

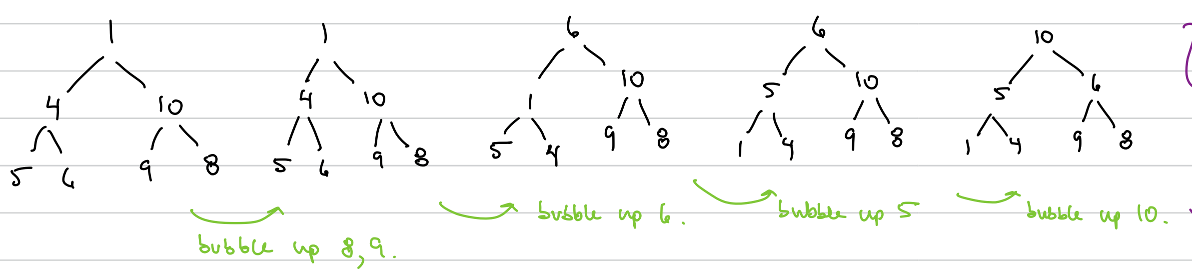
Substitute: $T(n) \leq T(n/6) + T(3n/4) + cn$

$$\leq \frac{dn}{6} + \frac{3dn}{4} + cn = dn\left(\frac{1}{6} + \frac{3}{4}\right) + cn$$

$$= dn\left(\frac{11}{12}\right) + cn = dn + cn - \frac{dn}{12} \leq dn$$ as long as $d > 12c$

3a) Method 1) Incorrect.

ex.



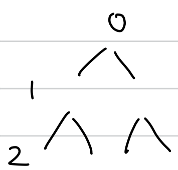
Incorrect final heap.

METHOD 2:) Correct. This is the bottom-up heap approach from class, starting at the last leaf instead of the last internal node. When bubble down is executed from a leaf, it simply exits since $i \nleq \lfloor A.\text{heapsize} \rfloor$.

Method 3) Correct. This is the iterative method for heap building, when we insert one element at a time, starting with $A[1]$. Recall that a heap insert is just a call to bubble-up from i.

3b) MiniMaxHeapInsert(A, k)

```
A.heapsize ++
A[A.heapsize] = k                 // insert new element in next 'leaf' position.
l = ⌊log₂ A.heapsize⌋.            // level of newly inserted node. (Root level is 0)
i = A.heapsize
if l is even                      // mini level
    if A[⌊i/2⌋] < A[i]            // too big
        Swap A[⌊i/2⌋] and A[i]    // swap with parent
        Max Bubble Up (A, ⌊i/2⌋)  // continue from max level
    else
        miniBubble Up (A, i)
else                              // max level
    if A[⌊i/2⌋] > A[i]            // too small
        Swap A[⌊i/2⌋] and A[i]    // swap with parent
        mini Bubble Up (A, ⌊i/2⌋) // continue from mini level.
    else
        MaxBubble Up (A, i)
```



Mini Bubble Up (A, i)

```
if i ≥ 4              // grandparent exists.
    p = ⌊i/2⌋
    g = ⌊p/2⌋.
    if A[i] < A[g]
        swap A[i], A[g]
        MiniBubble Up (A, g)
```

Max Bubble Up (A, i)

```
if i ≥ 4              // grandparent exists.
    p = ⌊i/2⌋
    g = ⌊p/2⌋
    if A[i] > A[g]
        Swap A[i], A[g]
        max Bubble Up (A, g)
```

3c) Student 1)
- Create array C: $O(n)$
- Bottom up heap building: $O(n)$ Total: $O(n \log n)$
- Heap Sort on C: $O(n \log n)$

Student 2)
- Total of n calls to delete max from B: $O(n \cdot \log n)$
- At most $2n$ comparisons: $O(n)$

Total: $O(n \log n)$

$\therefore$ both approaches have same worst-case runtime.

4a) Option 1) CORRECT: Step 2 will find the element of rank.
Step 3 results in a list of elements with rank $\leq \sqrt{n}$.
Step 4 sorts this list.
$\therefore$ Algorithm is correct.

Runtime:
- Step 1, 2: $O(n)$ since the Select alg. runs in time $O(n)$
- Step 3: for loop runs in time $O(n)$
- Step 4 runs in expected time $O(\sqrt{n} \log \sqrt{n})$
- Step 5 runs in time $O(\sqrt{n})$

Total: $O(n)$ expected runtime.

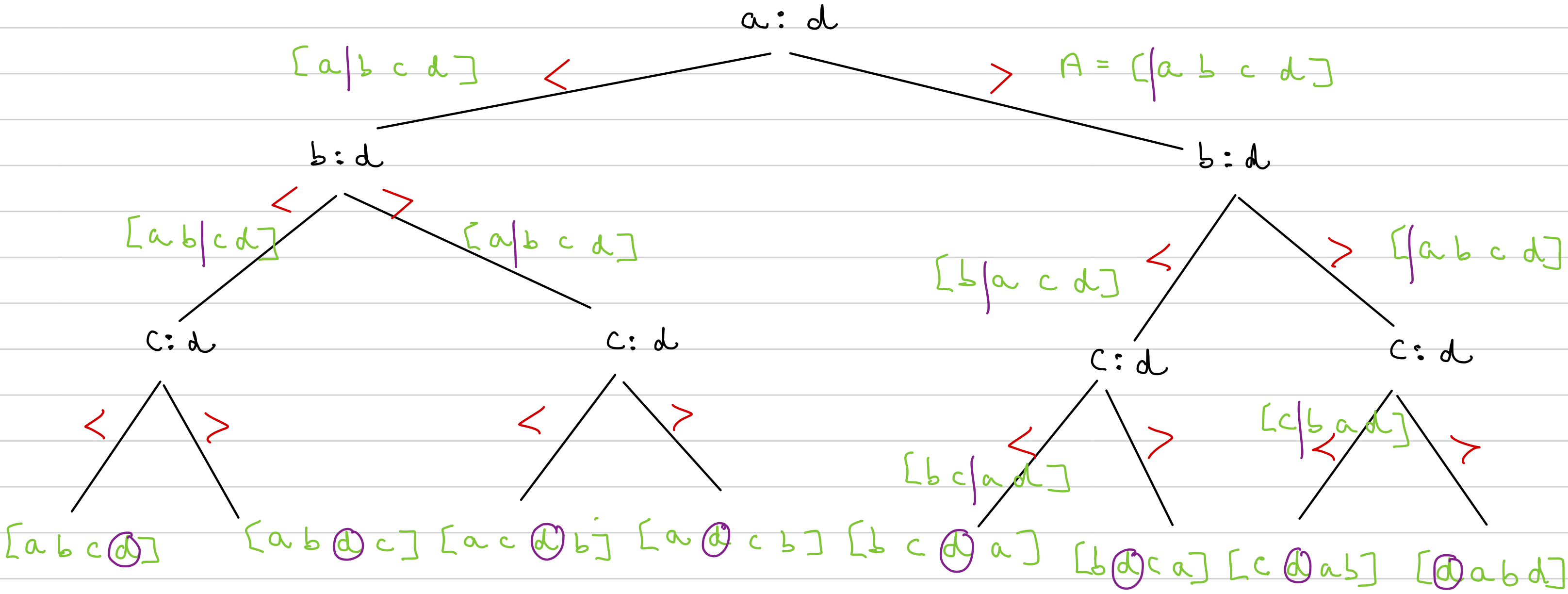
Option 2) INCORRECT: the uniformly distributed points are random, and therefore it is possible that the first bucket is empty. Running insertion sort on the first bucket will actually sort NOTHING if this happens.

Runtime: Distributing points runs in time $O(n)$
We expect $O(\sqrt{n})$ points in the first bucket, so insertion sort is expected to run in time $O(n)$.
Printing out the list is expected to run in time $O(\sqrt{n})$.
TOTAL: $O(n)$ expected time.
Doesn't ALWAYS return correct values.

Option 3) INCORRECT: Counting sort takes integers as input, not real #'s with an unlimited # of decimal places.

Runtime: If able to execute (only if input were ints), runtime is $O(n)$.

4b) Let A = [a b c d] Pivot: d.



4c) Radix sort simply runs a sequence of counting sorts.
Given plates of the form $P_1 P_2 P_3 P_4 P_5 P_6$:

1) Run counting sort on character in position 6 (P_6) where the input is in the range 0 ... 9. Time: $O(n+10) = O(n)$
2) Run counting sort on position 5 (P_5) where the input has 26 options which can be compared: A ... Z. Time: $O(n+26) = O(n)$

Repeat steps 1, 2 on P_4, P_3.
" " 1, 2 on P_2, P_1.

Runtime: $O(n)$.